

ECE 440 - Introduction to Computer Networks

Spring 2020

More about Web caching

- cache acts as both client and server
 - server for original requesting client
 - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

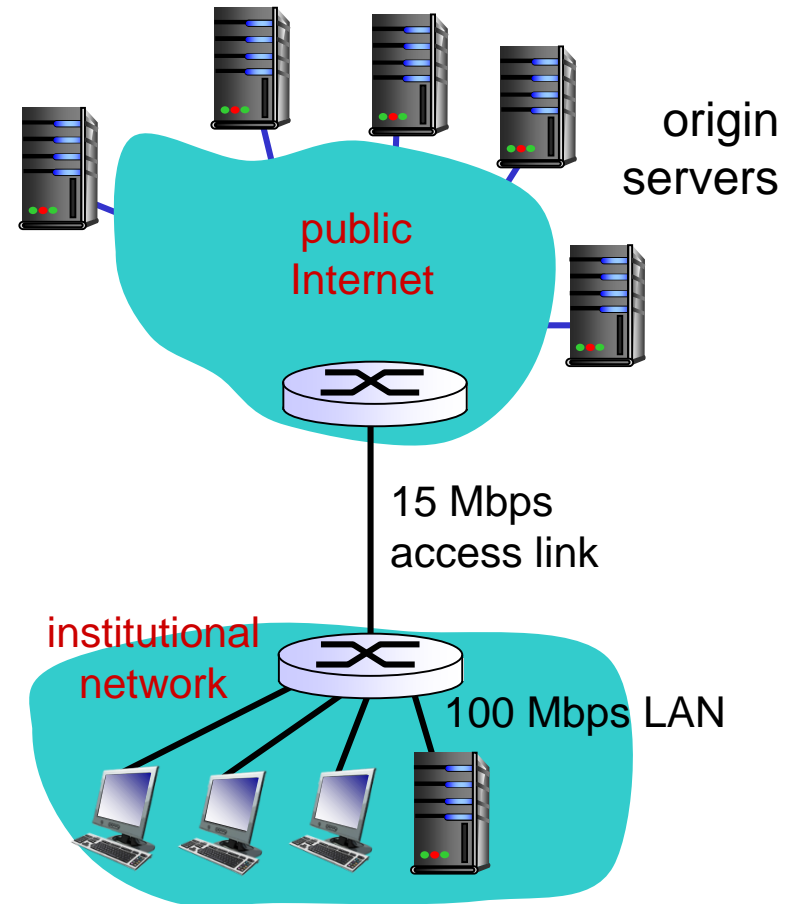
Caching example:

assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 100% *problem!*
- total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + msecs



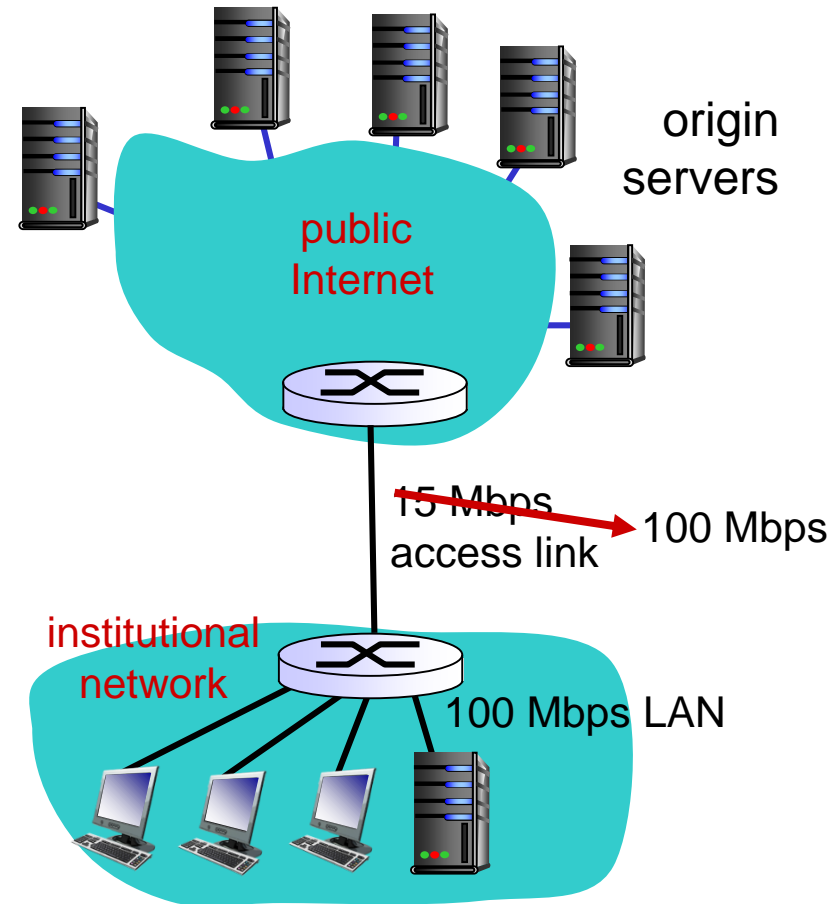
Caching example: fatter access link

assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~15 Mbps~~ → 100 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ → 9.9%
- total delay = Internet delay + access delay + LAN delay
= 2 sec + ~~minutes~~ → msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

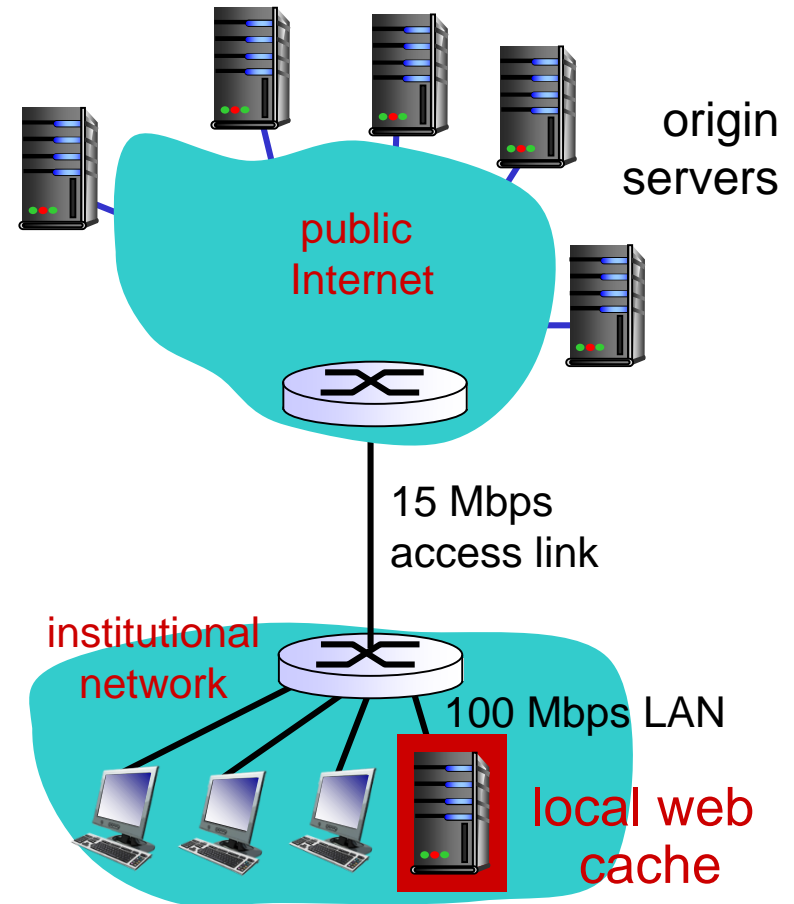
- avg object size: 1M bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

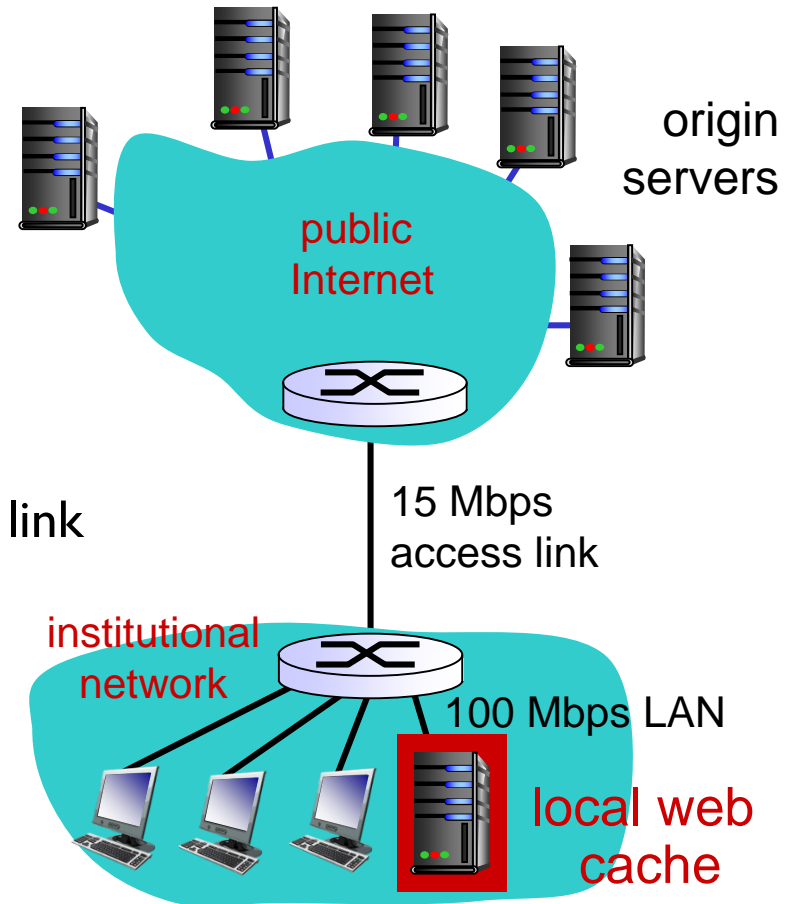
Cost: web cache (cheap!)



Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
 - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
 - 60% of requests use access link
- data rate to browsers over access link
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
 - utilization $= 0.9 / 15 = .06$
- total delay
 - $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
 - $= 0.6 (2.00) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$
 - less than with 100 Mbps link (and cheaper too!)



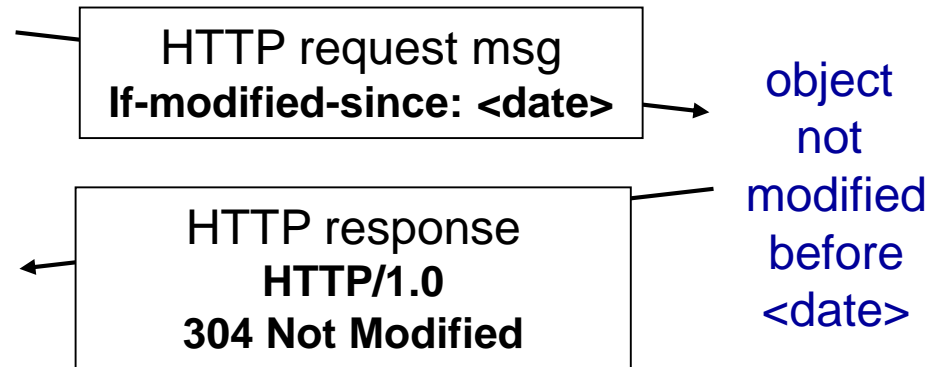
Conditional GET

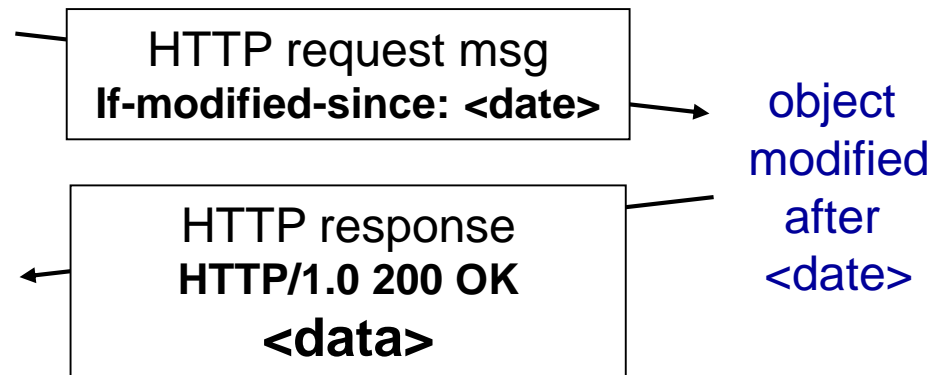
- **Goal:** don't send object if cache has up-to-date cached version
 - no object transmission delay
 - lower link utilization
- **cache:** specify date of cached copy in HTTP request
If-modified-since: <date>
- **server:** response contains no object if cached copy is up-to-date:
HTTP/1.0 304 Not Modified

client



server





Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

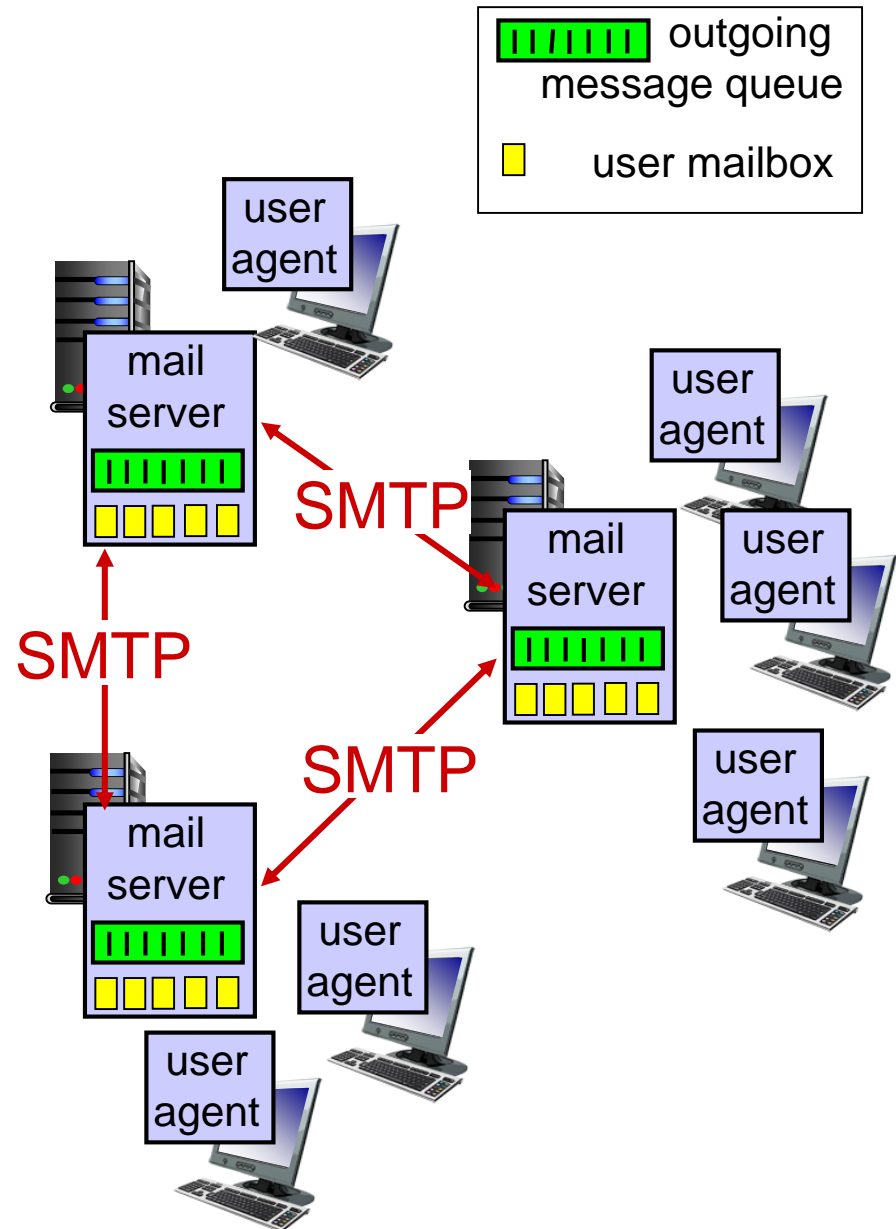
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP (simple mail transfer protocol)

User Agent

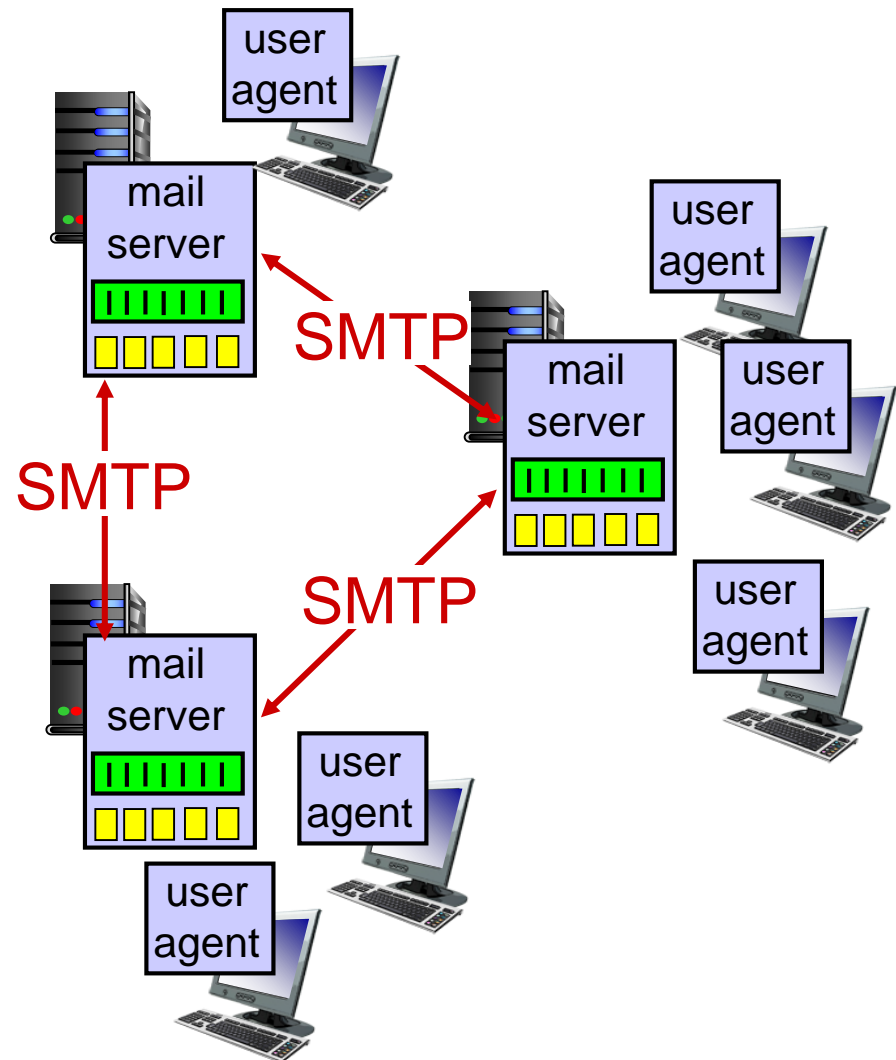
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
 - client: sending mail server
 - “server”: receiving mail server

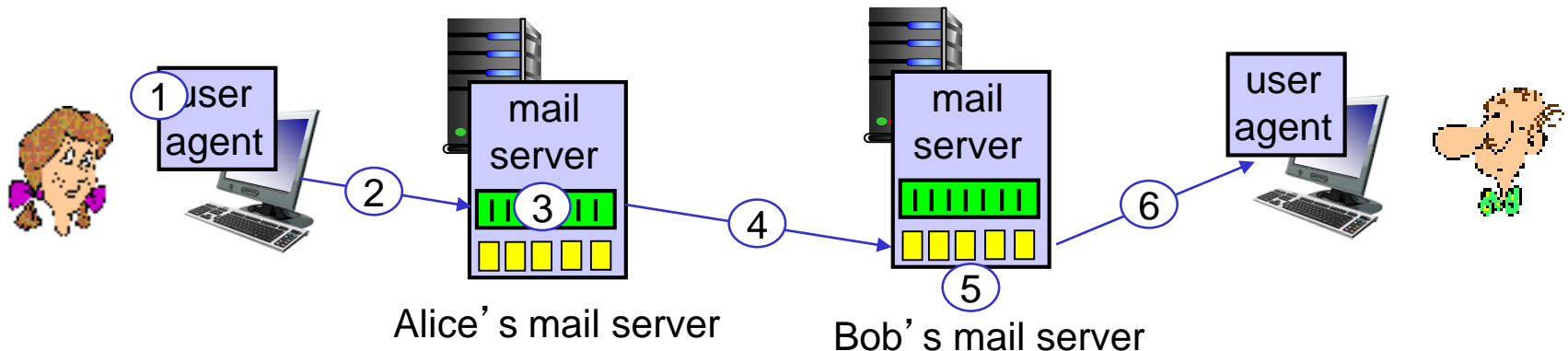


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
 - handshaking (greeting)
 - transfer of messages
 - closure
- command/response interaction (like HTTP)
 - **commands:** ASCII text
 - **response:** status code and phrase
- messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message “to”
`bob@someschool.edu`
- 2) Alice’s UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob’s mail server
- 4) SMTP client sends Alice’s message over the TCP connection
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

Mail message format

SMTP: protocol for exchanging email messages

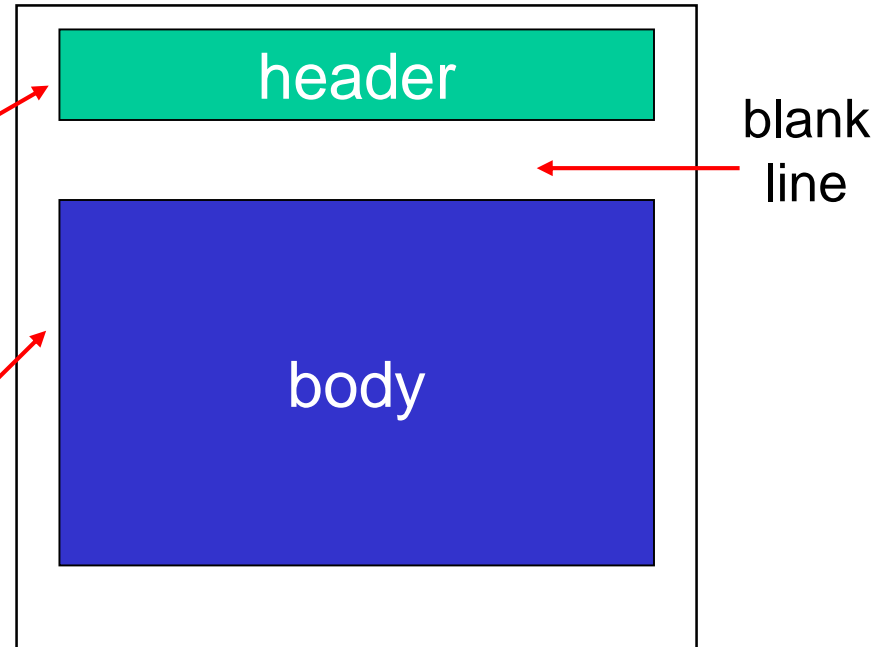
RFC 822: standard for text message format:

- header lines, e.g.,

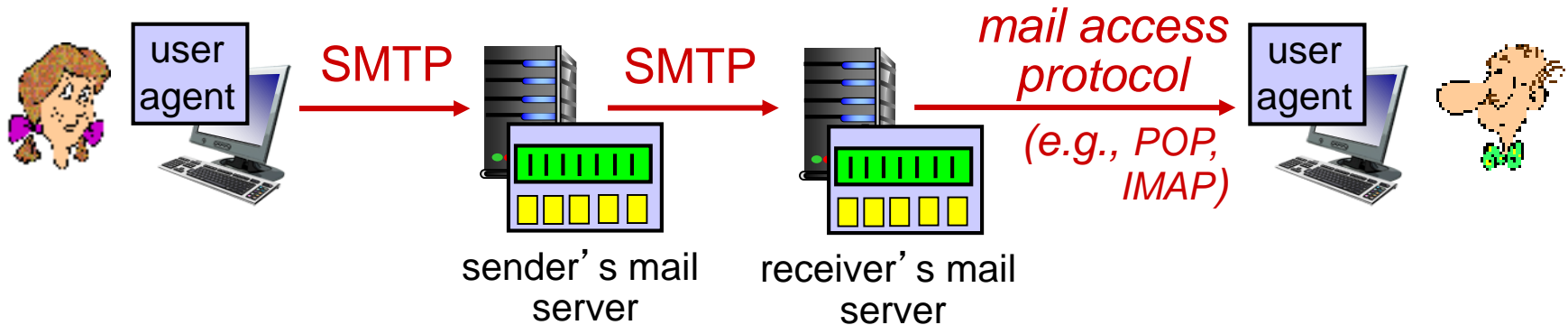
- To:
- From:
- Subject:

different from SMTP MAIL
FROM, RCPT TO:
commands!

- Body: the “message”
 - ASCII characters only



Mail access protocols



- **SMTP**: delivery/storage to receiver's server
- mail access protocol: retrieval from server
 - **POP**: Post Office Protocol [RFC 1939]: authorization, download
 - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
 - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase

- client commands:
 - **user**: declare username
 - **pass**: password
- server responses
 - **+OK**
 - **-ERR**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

transaction phase, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 “download and delete” mode
 - Bob cannot re-read e-mail if he changes client
- POP3 “download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
 - names of folders and mappings between message IDs and folder name

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., `www.yahoo.com` - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database*
implemented in hierarchy of many *name servers*
- *application-layer protocol*: hosts, name servers communicate to *resolve* names (address/name translation)
 - note: core Internet function, implemented as application-layer protocol
 - complexity at network's “edge”

DNS: services, structure

DNS services

- hostname to IP address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!